Word2Vec

Efficient model for learning Word Embeddings

Eric Daoud

Lab Acquisition, 02/12/16

Outline

Introduction

- The Word2Vec model is used for learning vector representations.
- Distributed representations of words in a vector space help learning algorithms to achieve better performance in NLP by grouping similar words (dimensionality reduction).
- It is based on Neural Language Model, with simplifications that make learning quicker.

Outline

1. Word Embeddings
   - What are they?
   - Why learn them?
   - Vector Space Models
2. Neural Language Model
   - Statistical Model of Language
   - Distributed Representations
   - Neural Networks
   - Feed Forward Neural Language Model (NNLM)
3. Word2Vec
   - Definition
   - Continuous Bag of Words (CBOW)
   - Continuous Skip-gram Model

Word Embeddings

What are they ?

- A word embedding[1] $W : words \to \mathbb{R}^n$ is a parametrized function mapping words to high-dimensional vectors.
- For instance: $W(\text{"}cat\text{"}) = (0.2, -0.4, 0.7 \ldots)$
- The function is typically a lookup table, parametrized by a matrix θ, with a row for each word: $W_\theta(w_n) = \theta_n$
- W is initialized to have random vectors for each word. It learns to have meaningful vectors in order to perform some task.

[1] http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

Outline

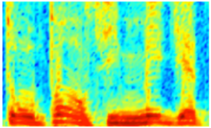
Word Embeddings

Why Learn Them ?

Because text data is sparse !

- Words are just 1's and 0's among the whole vocabulary
- A computer has no idea that “cat” is similar to “dog”
- We must find a way to learn how to group similar items together

AUDIO



Audio Spectrogram

DENSE

IMAGES

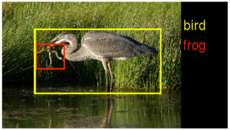


Image pixels

DENSE

TEXT

0	0	0	0.2	0	0.7	0	0	0	...	...

Word, context, or document vectors

SPARSE

Outline

1. Word Embeddings
   - What are they?
   - Why learn them?
   - Vector Space Models
2. Neural Language Model
   - Statistical Model of Language
   - Distributed Representations
   - Neural Networks
   - Feed Forward Neural Language Model (NNLM)
3. Word2Vec
   - Definition
   - Continuous Bag of Words (CBOW)
   - Continuous Skip-gram Model

- Vector space models embed words in a continuous vector space where semantically similar words are mapped to nearby points.
- Distributional Hypothesis: Words that appear in the same contexts share semantic meaning.
- Two different approaches that leverage this principle:
  1. Count-based methods (e.g. Latent Semantic Analysis)
  2. Predictive methods (e.g. Neural Probabilistic Language Models)

Outline

Statistical Model of Language

A statistical model of language can be represented by the conditional probability of the next word given all the previous ones: $P(w_t|context) \forall t \in V$:

$$\hat{P}(W_1^T) = \prod_{t=1}^{T} \hat{P}(w_t|w_1^{t-1}) \tag{1}$$

Outline

Distributed Representations

Perform an efficient dimensionality reduction by:

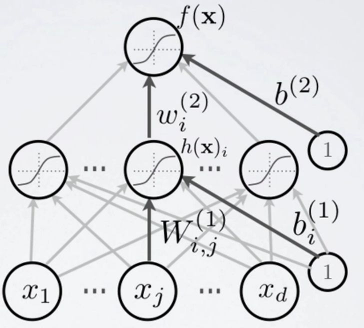
- Associate with each word in the vocabulary a word feature vector (which size is much smaller than the vocabulary length)
- Express the joint probability function of word sequences in terms of the feature vectors
- Learn simultaneously the word feature vectors and the parameters of that probability function

Learn the function parameters by maximizing the log-likelihood of the training data: $\hat{\theta}_{MLE} = argmax_{\theta} \sum_{i=1}^{n} f(x_i|\theta)$

Outline

Single hidden layer Neural Network

- Hidden layer pre-activation: $a(x) = b^{(1)} + W^{(1)}x$
- Hidden layer activation: $h(x) = g(a(x))$
- Output layer activation: $f(x) = o(b^{(2)} + w^{(2)^T} h^{(1)} x)$



For multi-class classification:

- We need multiple outputs (1 output per class)
- We want to estimate the conditional probability $p(y = c|x)$

We use the softmax activation function:

$$o(a) = softmax(a) = \left[\frac{exp(a_1)}{\sum_c exp(a_c)} \cdots \frac{exp(a_c)}{\sum_c exp(a_c)}\right] \tag{2}$$

Goal

Minimize the loss function.

Backpropagation algorithm:

1. Phase 1: Propagation
   1. Forward propagation that generates the output activations
   2. Backward propagation to generate the deltas
2. Phase 2: Weight update For each weight:
   1. Multiply its output delta and input activation to get the gradient
   2. Subtract a ratio from the gradient of the weight

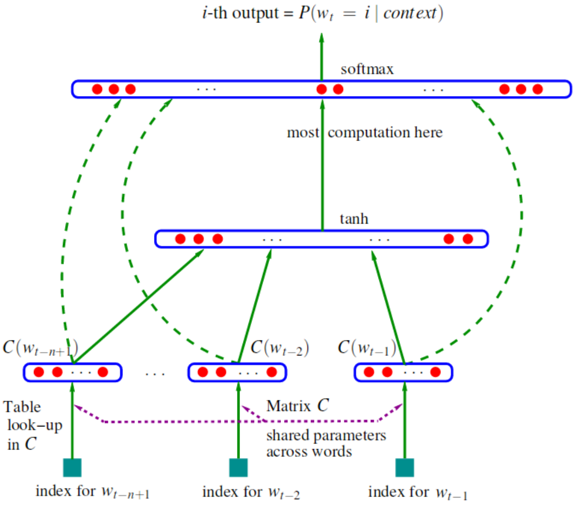
Outline

Feed Forward Neural Language Model (NNLM)[2]

- Training set: sequence $w_1, \ldots, w_T$ of words $w_T \in V$
- Objective: learn a good model $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t | w_t^{t-1})$ in the sense that it gives high out-of-sample likelihood.
- This function is split in two parts:
  1. A mapping C from any element i of V to a real vector $C(i) \in \mathbb{R}^m$.
  2. The probability function over words, expressed with C. A function g maps an input sequence of feature vectors for words in context to a conditional probability distribution over words in V for the next word w_t.

[2] http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

Feed Forward Neural Language Model (NNLM)



Feed Forward Neural Language Model (NNLM)

The function g can be implemented by a feed-forward neural network with parameters ω. The overall parameter set is $\theta = (C, \omega)$.

- Training is achieved by looking for θ that maximizes the training corpus penalized log-likelihood:

$$L = \frac{1}{T} \sum_t log\ f(w_t, w_{t-1}, \ldots, w_{t-n+1}; \theta) + R(\theta) \tag{3}$$

- The neural network computes the following function, with a softmax output layer:

$$\hat{P}(w_t | w_{t-1}, \ldots, w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{4}$$

- The y_i are the non normalized log-probabilities for each output word i:

$$y = tanh\ (b + Wx) \tag{5}$$

- x is the word features layer, as concatenation of the input word features from the matrix C:

$$x = (C(w_{t-1}), C(w_{t-2}), \ldots, C(w_{t-n+1})) \tag{6}$$

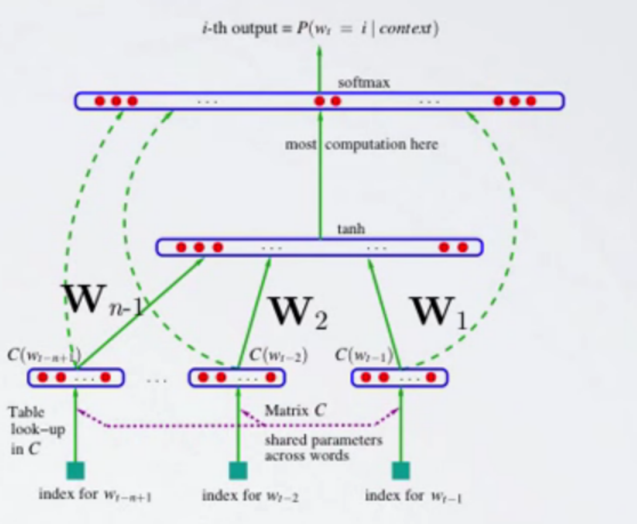
- After presenting the t-th word of the training corpus:

$$\theta \leftarrow \theta + \alpha \frac{\partial log \hat{P}(w_t | w_{t-1}, \ldots, w_{t-n+1})}{\partial \theta} \tag{7}$$

- Let's note $\nabla_{a(x)} l$ the gradient for the linear activation of the hidden layer. The gradient w.r.t $C(w)$ for any w is:

$$\nabla_{C(w)} l = \sum_{i=1}^{n-1} 1_{(w_{t-i}=w)} W_i^T \nabla_{a(x)} l \tag{8}$$

Feed Forward Neural Language Model (NNLM)



Feed Forward Neural Language Model (NNLM)

Example: [“the”, “dog”, “and”, “the”, “cat”]

- Loss function: $l = -log\ p\left(“cat"|“the", “dog", “and", “the"\right)$
- Assuming the following words have respective indexes 21, 3, 14, 21 and ? in our C matrix.
- Update the following representations:
  - $\nabla_{C(3)} l = W_3^T \nabla_{a(x)} l$
  - $\nabla_{C(14)} l = W_3^T \nabla_{a(x)} l$
  - $\nabla_{C(21)} l = W_1^T \nabla_{a(x)} l + W_4^T \nabla_{a(x)} l$
- $\nabla_{C(w)} l = 0$ for all other words w

Feed Forward Neural Language Model (NNLM)

Evaluation

We use the perplexity, which is the exponential of the average negative log-likelihood.

- The smaller the value is, the better
- Evaluation on Brown Corpus:
  - n-gram model: 321
  - neural network language model: 276
  - neural network + n-gram: 252

Feed Forward Neural Language Model (NNLM)

Issue

Such model is computationally expensive.

- Hierarchical Softmax helps training faster
- Newer log-linear models[3] have been developed !

[3] https://arxiv.org/pdf/1301.3781.pdf

Outline

Word2Vec

Definition

Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text.

The goal is to learn distributed representations of words that try to minimize computational complexity.

- Most of the complexity is caused by the non-linear hidden layer in the model.
- Simpler models might not be able to represent the data as precisely as neural networks, but can be trained on more data efficiently.

Word2Vec

It comes in two flavors:

1. Continuous Bag-of-Words model (CBOW)
2. Skip-Gram model

These models are algorithmically similar, except that:

- CBOW predicts target words from source context words
- Skip-gram does the opposite and predicts source-context words from target words

Outline

Continuous Bag of Words (CBOW)

Similar to the feedforward NNLM except that:

- The non linear hidden layer is removed
- All words get projected into the same position (their vectors are averaged)
- Bag of words architecture, as the order of words in history does not influence the projection
- We also use words from the future

Result

Best performance was using 4 future and 4 history words at the input, where the training criterion is trying to correctly classify the current (middle) word.

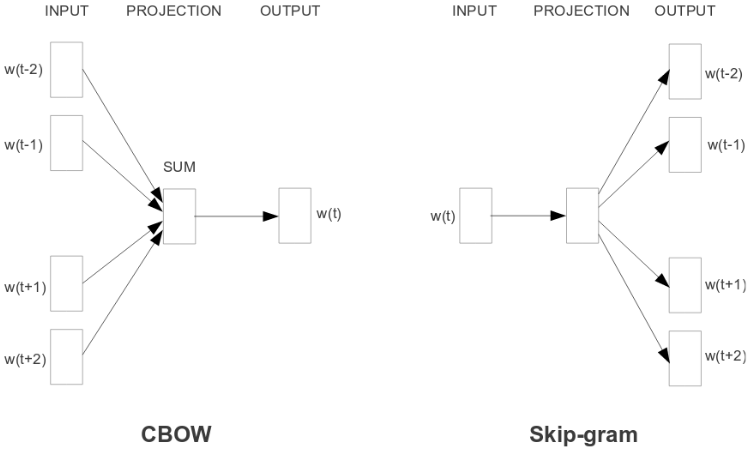
Outline

Continuous Skip-gram Model

Similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence.

- We use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word
- Increasing the range improves quality of the resulting word vectors, but also increases the computational complexity.

Comparison



INPUT
PROJECTION
OUTPUT
w(t-2)
w(t-1)
SUM
w(t)
w(t+1)
w(t+2)
CBOW
INPUT
PROJECTION
OUTPUT
w(t)
w(t-2)
w(t-1)
w(t+1)
w(t+2)
Skip-gram

Training

NNLM training issue

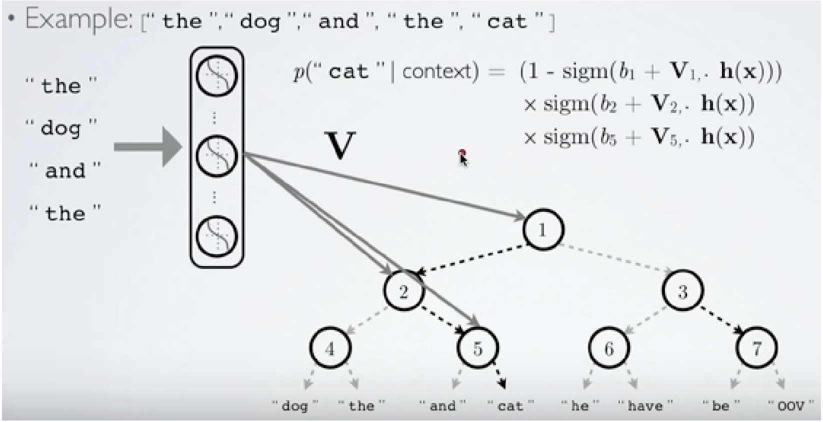
Training NNLM is impractical because the cost of computing $\nabla log\ p\left(w_O|w_I\right)$ is proportional to the number of words in the vocabulary, which is very large.

Two improvements[4]:

1. Hierarchical Softmax: Uses a binary tree representation of the output layer with the W words as its leaves and, for each node, explicitly represents the relative probabilities of its child nodes. Complexity drops to $log_2(W)$.
2. Negative Sampling: Simplified version of Noise Constrative Divergence, which states that a good model should be able to differentiate data from noise by means of logistic regression.

[4] https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Hierarchical Softmax



Negative Sampling

The objective for each example is to maximize:

$$J_{NEG} = log\ Q_\theta(D = 1|w_t, h) + k\ \mathbb{E}_{\tilde{w}\ P_{noise}}\left[log\ Q_\theta(D = 0|\tilde{w}, h)\right] \qquad (9)$$

Where $Q_\theta(D = 1|w, h)$ is the binary logistic regression probability under the model of seeing the word w in the context h in the dataset D.

- In practice, we approximate the expectation by drawing k constrative words from the noise distribution.
- This objective is maximized when the model assigns high probabilities to the real words, and low probabilities to noise words.
- Computationally it is especially appealing because computing the loss function now scales only with the number of noise words that we select, and not all words in the vocabulary.

Skip-gram Model Example[5]

Let's consider the dataset: "the quick brown fox jumped over the lazy dog".

- The context is defined as the window of words to the left and to the right of a target word.
- A window of size 1 brings: ([the, brown], quick), ([quick, fox], brown), . . . of (context, target) pairs.
- Skip gram inverts contexts and targets, and tries to predict each context word from its target word. The task becomes to predict "the" and "brown" from "quick", . . .
- Then the dataset becomes: (quick, the), (quick, brown), (brown, quick), . . . of (input, output) pairs.

[5] https://www.tensorflow.org/versions/r0.12/tutorials/word2vec/index.html

Skip-gram model Example

Let's imagine at training step t we observe the first training case above, where the goal is to predict "the" from "quick".

- We select num_noise number of noisy (contrastive) examples by drawing from some noise distribution, typically the unigram distribution.
- Say $num\_noise = 1$, and we draw "sheep" as a noisy example:

$$J_{NEG}^{t} = log(\ Q_\theta(D = 1|the, quick)) + log(\ Q_\theta(D = 0|sheep, quick)) \tag{10}$$

- We make an update to the embedding parameters θ by deriving the gradient of the loss with respect to θ:

$$\theta \leftarrow \theta + \alpha \frac{\partial}{\partial \theta} J_{NEG} \tag{11}$$

Conclusion

- Word2Vec is less able to capture word order, but much more efficient at training
- Skip-gram is prefered in the case of larger datasets
- Meta parameters have to be tuned properly in order to find the best mixture, leading to good results
- Works with raw text, but can be applied to other topics, like Recommender Systems ! e.g. predict next item in basket, or in navigation given the previous items considered.